Quality-Driven Synthesis of Embedded Multi-Mode Control Systems

Samii, Soheil; Eles, Petru; Peng, Zebo; Cervin, Anton

2009

Quality-Driven Synthesis of Embedded Multi-Mode Control Systems

Soheil Samii
Dept. of Computer and Information Science
Linköping University
Linköping, Sweden
sohsa@ida.liu.se

Petru Eles
Dept. of Computer and Information Science
Linköping University
Linköping, Sweden
petel@ida.liu.se

Zebo Peng
Dept. of Computer and Information Science
Linköping University
Linköping, Sweden
zpe@ida.liu.se

Anton Cervin
Dept. of Automatic Control
Lund University
Lund, Sweden
anton@control.lth.se

ABSTRACT

At runtime, an embedded control system can switch between alternative functional modes. In each mode, the system operates by using a schedule and controllers that exploit the available computation and communication resources to optimize the control performance in the running mode. The number of modes is usually exponential in the number of control loops, which means that all controllers and schedules cannot be produced in affordable design-time and stored in memory. This paper addresses synthesis of multi-mode embedded control systems. Our contribution is a method that trades control quality with optimization time, and that efficiently selects the schedules and controllers to be synthesized and stored in memory.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: *process control systems, real-time and embedded systems*; D.4.1 [**Operating Systems**]: Process Management—*scheduling*; J.6 [**Computer-Aided Engineering**]: *computer-aided design*; J.7 [**Computers in Other Systems**]: *industrial control*

General Terms

Algorithms, Design, Performance, Theory

Keywords

control performance, embedded control, multi-mode systems

1. INTRODUCTION AND RELATED WORK

In systems that control several physical plants, mode changes occur at runtime either as responses to external events or at predetermined moments in time. In an operation mode, the system controls a subset of the plants by executing several control applications concurrently—one for each controlled plant in the mode. The application tasks execute periodically (reading sensors, computing control signals, and writing to actuators) on a platform comprising several computation nodes connected to a bus. Such systems have complex timing behavior that leads to bad control performance if not taken into account during controller design [10]. To achieve good control performance in a certain mode, system scheduling must be integrated with controller design (periods and control laws). A mode change means that some of the running control loops are deactivated, some are activated, or both. This leads to a change in the execution and communication demand and, consequently, a new schedule and new controllers must be used to achieve best possible control performance by an efficient use of the resources.

DAC 2009, July 26–31, 2009, San Francisco, California, USA.

Control–scheduling co-design methods in the literature focus on single-mode control systems. Sampling-period optimization for mono-processor systems with multiple control loops and priority-based scheduling was first studied by Seto et al. [9]. Bini and Cervin improved their work by considering controller delays in the optimization process [2]. Rehbinder and Sanfridson proposed optimal control strategies for static scheduling of controllers on a mono-processor system [7]. In our previous work, we proposed a framework [8] for the scheduling and synthesis of controllers on distributed execution platforms.

The contribution of this paper is a scheduling and synthesis method for embedded multi-mode control systems. The number of modes to be considered in the synthesis is usually exponential in the number of control loops and leads to two problems: (1) all modes cannot be synthesized in affordable time, and (2) all synthesized controllers and schedules cannot be stored in memory. With the objective of optimizing the control performance in a multi-mode control system, we address these problems by a limited exploration of the set of modes, followed by a selection of the produced schedules and controllers to store in memory.

2. MULTI-MODE SYSTEMS

Given is a set of plants $\mathbf{P}$, indexed by $\mathcal{I}_{\mathbf{P}}$, where each plant P_i ($i \in \mathcal{I}_{\mathbf{P}}$) is described by a continuous-time linear model [11]

$$\dot{\boldsymbol{x}}_i = A_i \boldsymbol{x}_i + B_i \boldsymbol{u}_i + \boldsymbol{v}_i, \;\; \boldsymbol{y}_i = C_i \boldsymbol{x}_i + \boldsymbol{e}_i. \qquad (1)$$

The vectors $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the plant state and controlled input, respectively, and the vector $\boldsymbol{v}_i$ models plant disturbance as a continuous-time white-noise process with intensity R_{1i}. The continuous-time output $\boldsymbol{y}_i$ is measured and sampled periodically and is used to produce the control signal $\boldsymbol{u}_i$. The measurement noise $\boldsymbol{e}_i$ in the output is modeled as a discrete-time white-noise process with variance R_{2i}. The control signal is updated at discrete time instants and is held constant between two updates (zero-order hold [11]). As an example, let us consider three inverted pendulums $\mathbf{P} = \{P_1, P_2, P_3\}$. Each pendulum P_i ($i \in \mathcal{I}_{\mathbf{P}} = \{1, 2, 3\}$) is modeled according to Equation 1, with $A_i = \begin{bmatrix} 0 & 1 \\ g/l_i & 0 \end{bmatrix}$, $B_i = \begin{bmatrix} 0 & g/l_i \end{bmatrix}^{\mathsf{T}}$, and $C_i = \begin{bmatrix} 1 & 0 \end{bmatrix}$, where $g \approx 9.81$ m/s^2 and l_i are the gravitational constant and length of pendulum P_i, respectively (all pendulums have equal length, $l_i = 0.2$ m). For the plant disturbance and measurement noise, we have $R_{1i} = B_i B_i^{\mathsf{T}}$ and $R_{2i} = 0.1$.

The execution platform comprises a set of nodes $\mathbf{N}$, indexed by $\mathcal{I}_{\mathbf{N}}$, which are connected by a communication controller to a bus; common communication protocols in automotive systems are TTP [5], FlexRay [4], and CAN [3], which all are supported by our framework. On the execution platform runs a set of applications $\mathbf{\Lambda}$, indexed by the set $\mathcal{I}_{\mathbf{\Lambda}} = \mathcal{I}_{\mathbf{P}}$. Application $\Lambda_i \in \mathbf{\Lambda}$ ($i \in \mathcal{I}_{\mathbf{\Lambda}}$) controls plant P_i and is modeled as a directed acyclic graph $\Lambda_i = (\mathbf{T}_i, \mathbf{\Gamma}_i)$, where the nodes $\mathbf{T}_i$, indexed by $\mathcal{I}_i$, represent computation tasks and the edges $\mathbf{\Gamma}_i \subset \mathbf{T}_i \times \mathbf{T}_i$ represent data dependencies between tasks. Figure 1 shows two nodes $\mathbf{N} = \{N_1, N_2\}$ connected to a bus. Three applications $\mathbf{\Lambda} = \{\Lambda_1, \Lambda_2, \Lambda_3\}$ are mapped to the nodes and control

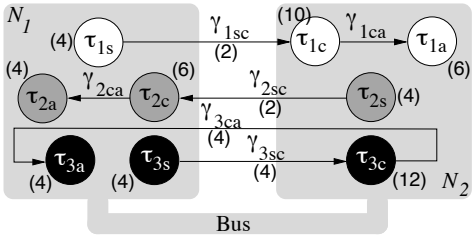

Figure 1: Motivational example

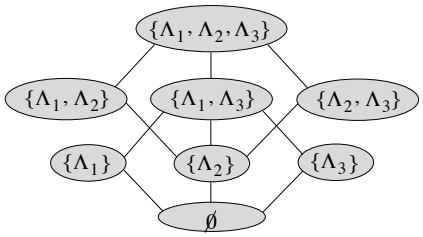

Figure 2: Hasse diagram of modes

Table 1: Individual control costs

Mode $\mathbf{M}$	$J_1^{\mathbf{M}}$	$J_2^{\mathbf{M}}$	$J_3^{\mathbf{M}}$
$\{\Lambda_1, \Lambda_2, \Lambda_3\}$	2.98	1.19	2.24
$\{\Lambda_1, \Lambda_2\}$	1.60	1.42	-
$\{\Lambda_1, \Lambda_3\}$	1.60	-	1.95
$\{\Lambda_2, \Lambda_3\}$	-	1.36	1.95
$\{\Lambda_1\}$	1.60	-	-
$\{\Lambda_2\}$	-	1.14	-
$\{\Lambda_3\}$	-	-	1.87

the three pendulums P_1, P_2, and P_3 in the example before. For this example, the task set of application Λ_i is $\mathbf{T}_i = \{\tau_{is}, \tau_{ic}, \tau_{ia}\}$ ($\mathcal{I}_i = \{s, c, a\}$). The data dependencies are given by the edges $\mathbf{\Gamma}_i = \{\gamma_{isc} = (\tau_{is}, \tau_{ic}), \gamma_{ica} = (\tau_{ic}, \tau_{ia})\}$. The sensor task is τ_{is}, the controller task is τ_{ic}, and the actuator task is τ_{ia}.

The system can run in different operation modes, where each mode is characterized by a set of running applications. A mode is a subset $\mathbf{M} \subseteq \mathbf{\Lambda}$, indexed by $\mathcal{I}_{\mathbf{M}} \subseteq \mathcal{I}_{\mathbf{\Lambda}} = \mathcal{I}_{\mathbf{P}}$, that contains the applications that are running in that mode. The complete set of modes is $\mathcal{M} = 2^{\mathbf{\Lambda}}$ and its cardinality is $|\mathcal{M}| = 2^{|\mathbf{\Lambda}|}$. The set of modes $\mathcal{M}$ is a partially ordered set under the subset relation $\subset$. For our example with the three applications in Figure 1, we show in Figure 2 the Hasse diagram of the partially ordered set of modes $\mathcal{M} = 2^{\mathbf{\Lambda}} = 2^{\{\Lambda_1, \Lambda_2, \Lambda_3\}}$. A mode $\mathbf{M}' \in \mathcal{M}$ is called a *submode* of $\mathbf{M}$ if $\mathbf{M}' \subset \mathbf{M}$. Mode $\mathbf{M}$ is called a *supermode* of mode $\mathbf{M}'$. We define the set of submodes of $\mathbf{M} \in \mathcal{M}$ as $\underline{\mathcal{M}}(\mathbf{M}) = \{\mathbf{M}' \in \mathcal{M} : \mathbf{M}' \subset \mathbf{M}\}$. Similarly, the set of supermodes of $\mathbf{M}$ is denoted $\overline{\mathcal{M}}(\mathbf{M})$. The idle mode is the empty set $\emptyset$, which indicates that the system is inactive, whereas the mode $\mathbf{\Lambda}$ indicates that all applications are running. It can be the case that certain modes do not occur at runtime—for example, because certain plants are never controlled concurrently. Let us therefore introduce the set of *functional* modes $\mathcal{M}^{\text{func}} \subseteq \mathcal{M}$ that can occur during execution. Modes $\mathcal{M}^{\text{virt}} = \mathcal{M} \setminus \mathcal{M}^{\text{func}}$ do not occur at runtime and are therefore called *virtual* modes. In a given mode $\mathbf{M} \in \mathcal{M}$, an application $\Lambda_i \in \mathbf{M}$ ($i \in \mathcal{I}_{\mathbf{M}}$) releases jobs for execution periodically with the period $h_i^{\mathbf{M}}$. Job q of task τ_{ij} is denoted $\tau_{ij}^{(q)}$. For a message $\gamma_{ijk} = (\tau_{ij}, \tau_{ik}) \in \mathbf{\Gamma}_i$, the message instance produced by job $\tau_{ij}^{(q)}$ is denoted $\gamma_{ijk}^{(q)}$. An edge $\gamma_{ijk} = (\tau_{ij}, \tau_{ik}) \in \mathbf{\Gamma}_i$ means that the earliest start time of a job $\tau_{ik}^{(q)}$ is when $\tau_{ij}^{(q)}$ has completed its execution and the produced data (i.e., $\gamma_{ijk}^{(q)}$) has been communicated to $\tau_{ik}^{(q)}$. We define the hyperperiod $h_{\mathbf{M}}$ of $\mathbf{M}$ as the least common multiple of the periods $\{h_i^{\mathbf{M}}\}_{i \in \mathcal{I}_{\mathbf{M}}}$.

Each task is mapped to a node; in Figure 1, task τ_{1s} is mapped to N_1 and τ_{1c} is mapped to N_2. A message between tasks that are mapped to different nodes is sent on the bus. For a message γ_{ijk} between two nodes, we denote with c_{ijk} the communication time when there are no conflicts on the bus. We model the execution time of task τ_{ij} as a stochastic variable c_{ij} with probability function $\xi_{c_{ij}}$. We assume that the execution time of τ_{ij} is bounded by given best-case and worst-case execution times. In Figure 1, the execution times (constant in this example) and communication times for the tasks and messages are given in milliseconds in parentheses. We support static-cyclic and priority-based scheduling of tasks and messages [5]. In static-cyclic scheduling, the period of the schedule is the hyperperiod $h_{\mathbf{M}}$ of the applications in the running mode $\mathbf{M}$. The schedule determines the start times of the jobs and message instances that are released within a hyperperiod of the mode. Further, the schedule must satisfy precedence constraints, which are given by the data dependencies, and account for the worst-case execution times of the tasks and the communication times of the messages. For preemptive priority-based scheduling, the tasks and messages are scheduled at runtime based on fixed priorities that are decided at design time.

As an example of a static-cyclic schedule, let us consider the system in Figure 1 and the schedule in Figure 3. All times are given in milliseconds in the discussion that follows. The schedule is constructed for mode $\mathbf{M} = \mathbf{\Lambda}$ (i.e., for the mode in which all three applications are running concurrently) and for the periods $h_1^{\mathbf{M}} = 40$, $h_2^{\mathbf{M}} = 20$, and $h_3^{\mathbf{M}} = 40$. The period of the schedule is $h_{\mathbf{M}} = 40$ (the hyperperiod of $\mathbf{M}$). Considering the periods of the applications, the jobs to be scheduled on node N_1 are $\tau_{1s}^{(1)}$, $\tau_{2c}^{(1)}$, $\tau_{2c}^{(2)}$, $\tau_{2a}^{(1)}$, $\tau_{2a}^{(2)}$, $\tau_{3s}^{(1)}$, and $\tau_{3a}^{(1)}$. The jobs on node N_2 are $\tau_{1c}^{(1)}$, $\tau_{1a}^{(1)}$, $\tau_{2s}^{(1)}$, $\tau_{2s}^{(2)}$, and $\tau_{3c}^{(1)}$. The message transmissions on the bus are $\gamma_{1sc}^{(1)}$, $\gamma_{2sc}^{(1)}$, $\gamma_{2sc}^{(2)}$, $\gamma_{3sc}^{(1)}$, and $\gamma_{3ca}^{(1)}$. The schedule in Figure 3 is shown with three rows for node N_1, the bus, and node N_2, respectively. The small boxes depict task executions and message transmissions. The white, grey, and black boxes show the execution of applications Λ_1, Λ_2, and Λ_3, respectively. Each box is labeled with an index that indicates a task or message, and with a number that specifies the job or message instance. The black box labeled $c(1)$ shows that the execution of job $\tau_{3c}^{(1)}$ on node N_2 starts at time 8 and completes at time 20. The grey box labeled $sc(2)$ shows the second message between the sensor and controller task of Λ_2.

An *implementation* of a mode $\mathbf{M} \in \mathcal{M}$ comprises the period $h_i^{\mathbf{M}}$ and control law $\boldsymbol{u}_i^{\mathbf{M}}$ of each control application $\Lambda_i \in \mathbf{M}$ ($i \in \mathcal{I}_{\mathbf{M}}$), and the schedule (or priorities) for the tasks and messages in the mode. We denote with $\mathsf{mem}_d^{\mathbf{M}}$ the memory consumption on node $N_d \in \mathbf{N}$ ($d \in \mathcal{I}_{\mathbf{N}}$) of the implementation of $\mathbf{M}$. An implementation of a mode can serve as an implementation of all its submodes. This means that the system can run in a submode $\mathbf{M}' \subset \mathbf{M}$ ($\mathbf{M}' \neq \emptyset$) with the same controllers (periods and control laws) and schedule as for mode $\mathbf{M}$—for example, by not running the applications in $\mathbf{M} \setminus \mathbf{M}'$ or by not writing to their outputs. However, to achieve better performance in mode $\mathbf{M}'$, a customized set of controllers and schedule for submode $\mathbf{M}'$ can exploit the available computation and communication resources that are now not used by the other applications $\mathbf{M} \setminus \mathbf{M}'$. To have a correct implementation of the whole system, for each functional mode $\mathbf{M} \in \mathcal{M}^{\text{func}}$, there must exist an implementation in memory, or there must exist an implementation of at least one of its supermodes. The set of implemented modes is $\mathcal{M}^{\text{impl}} \subseteq \mathcal{M} \setminus \{\emptyset\}$.

3. CONTROL QUALITY AND SYNTHESIS

The quality of a controller Λ_i for plant P_i (Equation 1) is given by the quadratic cost [11]

$$J_i = \lim_{T \to \infty} \frac{1}{T} \mathrm{E}\left\{ \int_0^T \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix}^{\mathsf{T}} Q_i \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix} dt \right\}. \qquad (2)$$

The matrix Q_i, which is given by the designer, is a positive semi-definite matrix with weights for the magnitude of the plant states and the control signals ($\mathrm{E}\{\cdot\}$ denotes the expected value of a stochastic variable). For a given sampling period h_i and a given constant sensor–actuator delay (i.e., the time elapsed between sampling the output $\boldsymbol{y}_i$ and updating the controlled input $\boldsymbol{u}_i$), it is possible to find the control law $\boldsymbol{u}_i$ that minimizes J_i [11]. The cost J_i of a controller is increased (its quality is

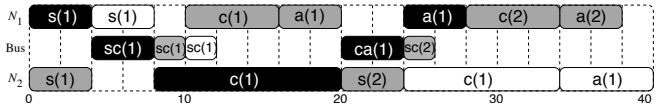

Figure 3: Schedule for mode $\{\Lambda_1, \Lambda_2, \Lambda_3\}$ with periods $h_1 = 40$, $h_2 = 20$, and $h_3 = 40$

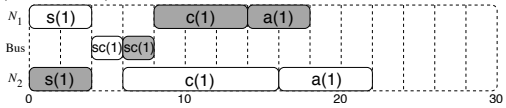

Figure 4: Schedule for mode $\{\Lambda_1, \Lambda_2\}$ with periods $h_1 = 30$ and $h_2 = 30$

degraded) if the sensor–actuator delay at runtime is different from what was assumed during control-law synthesis, or if this delay is not constant (i.e., there is jitter). Given the delay characteristics, we compute this cost J_i with Jitterbug [6].

We have developed a framework [8] to synthesize controllers and schedule their execution and communication for single-mode systems. In this paper, we use the framework to obtain an implementation of a certain mode $\mathbf{M} \in \mathcal{M} \setminus \{\emptyset\}$ such that the total control cost of the mode $J^{\mathbf{M}} = \sum_{i \in \mathcal{I}_{\mathbf{M}}} J_i^{\mathbf{M}}$ is minimized, where $J_i^{\mathbf{M}}$ is the cost of controller Λ_i. The produced mode implementation comprises the periods $h_i^{\mathbf{M}}$, control laws $\boldsymbol{u}_i^{\mathbf{M}}$, and system schedule (schedule table or priorities). We also obtain the memory consumption $\mathsf{mem}_d^{\mathbf{M}} \in \mathbb{N}$ of the implementation on each computation node $N_d \in \mathbf{N}$ ($d \in \mathcal{I}_{\mathbf{N}}$).

4. MOTIVATIONAL EXAMPLE

In this section, we shall highlight the two problems that are addressed in this paper: (1) the time complexity of the synthesis of embedded multi-mode control systems, and (2) the memory complexity of the storage of produced mode implementations. Let us consider our example in Section 2 with three applications $\mathbf{\Lambda} = \{\Lambda_1, \Lambda_2, \Lambda_3\}$ that control the three inverted pendulums $\mathbf{P} = \{P_1, P_2, P_3\}$. For the computation of the control cost in Equation 2, we have used $Q_i = \mathrm{diag}(C_i^{\mathsf{T}} C_i, 0.002)$ as the weight matrix for each inverted pendulum P_i.

By using our framework for controller scheduling and synthesis [8], we synthesized an implementation of mode $\mathbf{M}_{123} = \mathbf{\Lambda}$ with the periods $h_1^{\mathbf{M}_{123}} = 40$, $h_2^{\mathbf{M}_{123}} = 20$, and $h_3^{\mathbf{M}_{123}} = 40$ and the schedule in Figure 3. The outputs of the plants are sampled periodically without jitter (e.g., by some mechanism that stores the sampled data in buffers). The actuations of Λ_1 and Λ_3 finish at times 40 and 28, respectively. Because the schedule is periodic, the delay from sampling to actuation is 40 in each instance of Λ_1 (i.e., constant). Similarly, application Λ_3 has the constant sampling–actuation delay 28. Two instances are scheduled for application Λ_2. The first actuation finishes at time 20, whereas the second actuation finishes at time 38. By considering the sampling period 20, we note that the sampling–actuation delay of Λ_2 during periodic execution of the schedule is either 20 or 18. With this implementation, we obtained the individual control costs $J_1^{\mathbf{M}_{123}} = 2.98$, $J_2^{\mathbf{M}_{123}} = 1.19$, and $J_3^{\mathbf{M}_{123}} = 2.24$. Table 1 contains the individual controller costs of the modes considered in this section.

Let us now consider mode $\mathbf{M}_{12} = \{\Lambda_1, \Lambda_2\}$ in which Λ_3 is not executing. The system can operate in the new mode by using the schedule and control laws from mode $\mathbf{M}_{123}$. This can be done by not writing to the actuators of Λ_3 or by omitting the execution and communication of Λ_3. By using the implementation of mode $\mathbf{M}_{123}$, the overall control cost of mode $\mathbf{M}_{12}$ is $J_1^{\mathbf{M}_{123}} + J_2^{\mathbf{M}_{123}} = 2.98 + 1.19 = 4.17$. This cost can be reduced because, compared to mode $\mathbf{M}_{123}$, there is now more computation and communication power available for applications Λ_1 and Λ_2. Thus, it is worth investigating whether a better implementation (e.g., with reduced periods and delays) can be produced for mode $\mathbf{M}_{12}$. By running the synthesis of $\mathbf{M}_{12}$, we obtained an implementation with the periods $h_1^{\mathbf{M}_{12}} = h_2^{\mathbf{M}_{12}} = 30$ and the schedule in Figure 4. Note from the new schedule that both the period and delay of Λ_1 have been reduced. The costs of Λ_1 and Λ_2 with the new implementation are $J_1^{\mathbf{M}_{12}} = 1.60$ and $J_2^{\mathbf{M}_{12}} = 1.42$ (Table 1). The cost of Λ_1 is reduced significantly as a result of the reduction in sampling period and delay. The sampling period of Λ_2 is increased, which results in a small increase in the cost of Λ_2. This cost increase is accepted because it makes possible a significant quality improvement of Λ_1, which leads to a significant cost reduction of mode $\mathbf{M}_{12}$ to 3.02.

To achieve the best control performance, implementations of all functional modes have to be produced at design time. However, the number of modes is exponential in the number of control loops that run on the platform. Thus, even if some modes are functionally excluded, implementations of all possible functional modes cannot be produced in affordable time (except cases with small number of control loops). Let us consider that we can only run the synthesis of three modes at design time. Modes $\mathbf{M}_{123}$ and $\mathbf{M}_{12}$ have already been discussed. Considering the third mode to be $\mathbf{M}_{13}$, the synthesis resulted in the costs $J_1^{\mathbf{M}_{13}} = 1.60$ and $J_3^{\mathbf{M}_{13}} = 1.95$ (Table 1). When using the implementation of $\mathbf{M}_{123}$, the costs of Λ_1 and Λ_3 are 2.98 and 2.24, respectively. By running $\mathbf{M}_{13}$ with the new implementation thus leads to a significant improvement in control performance, compared to when using the implementation of $\mathbf{M}_{123}$. At runtime, the system has implementations of the modes $\mathcal{M}^{\mathrm{impl}} = \{\mathbf{M}_{123}, \mathbf{M}_{12}, \mathbf{M}_{13}\}$ in memory. For modes that are not implemented, the system chooses an implementation at runtime based on the three available implementations. For example, mode $\mathbf{M}_2 = \{\Lambda_2\}$ does not have an implementation but can run with the implementation of either $\mathbf{M}_{12}$ or $\mathbf{M}_{123}$. The cost of Λ_2 when running with the implementation of $\mathbf{M}_{12}$ is $J_2^{\mathbf{M}_{12}} = 1.42$, whereas it is $J_2^{\mathbf{M}_{123}} = 1.19$ for the implementation of $\mathbf{M}_{123}$. Thus, at runtime, the implementation of $\mathbf{M}_{123}$ is chosen to operate the system in mode $\mathbf{M}_2$.

We now consider that memory limitations in the platform imply that we can only store implementations of two modes out of the three modes in $\mathcal{M}^{\mathrm{impl}}$. We cannot use the implementation of $\mathbf{M}_{12}$ or $\mathbf{M}_{13}$ to run the system in mode $\mathbf{M}_{123}$ and thus we cannot remove its implementation. As we discussed in the beginning of this section, the total control cost when running mode $\mathbf{M}_{12}$ with the implementation of $\mathbf{M}_{123}$ is 4.17, compared to the total cost 3.02 when running with the implementation of $\mathbf{M}_{12}$. The implementation of $\mathbf{M}_{12}$ thus gives a total cost reduction of 1.15. If, on the other hand, $\mathbf{M}_{13}$ runs with the implementation of $\mathbf{M}_{123}$, the total cost is $J_1^{\mathbf{M}_{123}} + J_3^{\mathbf{M}_{123}} = 2.98 + 2.24 = 5.22$. If $\mathbf{M}_{13}$ runs with its produced implementation, the total cost is $J_1^{\mathbf{M}_{13}} + J_3^{\mathbf{M}_{13}} = 1.60 + 1.95 = 3.55$. This gives a total cost reduction of 1.67, which is better than the reduction obtained by the implementation of $\mathbf{M}_{12}$. Thus, in the presence of memory limitations, the implementations of $\mathbf{M}_{123}$ and $\mathbf{M}_{13}$ should be stored in memory to achieve the best control performance. In this discussion, we have assumed that $\mathbf{M}_{12}$ and $\mathbf{M}_{13}$ are equally important. However, if $\mathbf{M}_{12}$ occurs more frequently than $\mathbf{M}_{13}$, the cost improvement of the implementation of $\mathbf{M}_{12}$ becomes more significant. In this case, the best selection could be to exclude the implementation of $\mathbf{M}_{13}$ and store implementations of $\mathbf{M}_{123}$ and $\mathbf{M}_{12}$ in memory.

As the last example, let us consider that $\mathbf{M}_{123}$ is a virtual mode (i.e., it does not occur at runtime). Let $\mathcal{M}^{\mathrm{impl}} = \{\mathbf{M}_{12}, \mathbf{M}_{13}, \mathbf{M}_{23}\}$ be the set of modes with produced implementations. We have run the synthesis of mode $\mathbf{M}_{23} = \{\Lambda_2, \Lambda_3\}$ and obtained a total cost of 3.31. Let us assume that the three produced implementations of the modes in $\mathcal{M}^{\mathrm{impl}}$ cannot be all stored in memory. If the implementation of $\mathbf{M}_{23}$ is removed,

for example, there is no valid implementation of the functional mode $\mathbf{M}_{23}$ in memory. To solve this problem, we implement the virtual mode $\mathbf{M}_{123}$. Its implementation can be used to run the system in all functional modes—however, with degraded control performance. The available memory allows us to further store the implementation of one of the modes $\mathbf{M}_{12}$, $\mathbf{M}_{13}$, or $\mathbf{M}_{23}$. We choose the mode implementation that gives the largest cost reduction, compared to the implementation with $\mathbf{M}_{123}$. By looking in Table 1 and with the cost reductions in our discussions earlier in this example, we conclude that $\mathbf{M}_{13}$ gives the largest cost reduction among the modes in $\mathcal{M}^{\text{impl}}$. The best control performance under these tight memory constraints is achieved if the virtual mode $\mathbf{M}_{123}$ and functional mode $\mathbf{M}_{13}$ are implemented and stored in memory. This shows that memory limitations can lead to situations in which virtual modes must be implemented to cover a set of functional modes.

5. PROBLEM FORMULATION

In addition to the plants, control applications, and their mapping to an execution platform (Section 2), the inputs to the problem that we address are

- a set of functional modes[1] $\mathcal{M}^{\text{func}} \subseteq \mathcal{M}$ that can occur at runtime (the set of virtual modes is $\mathcal{M}^{\text{virt}} = \mathcal{M} \setminus \mathcal{M}^{\text{func}}$),
- a weight[2] $w_{\mathbf{M}} > 0$ for each mode $\mathbf{M} \in \mathcal{M}^{\text{func}}$ ($w_{\mathbf{M}} = 0$ for $\mathbf{M} \in \mathcal{M}^{\text{virt}}$), and
- the available memory in the platform, modeled as a memory limit $\mathsf{mem}_d^{\max} \in \mathbb{N}$ for each node N_d ($d \in \mathcal{I}_{\mathbf{N}}$).

The outputs are implementations of a set of modes $\mathcal{M}^{\text{impl}} \subseteq \mathcal{M} \setminus \{\emptyset\}$. For each functional mode $\mathbf{M} \in \mathcal{M}^{\text{func}} \setminus \{\emptyset\}$, there must exist an implementation (i.e., $\mathbf{M} \in \mathcal{M}^{\text{impl}}$) or a supermode implementation (i.e., $\mathcal{M}^{\text{impl}} \cap \overline{\mathcal{M}}(\mathbf{M}) \neq \emptyset$).

If $\mathbf{M} \in \mathcal{M}^{\text{impl}}$, the cost $J^{\mathbf{M}}$ is given from the scheduling and synthesis step that produces the implementation (Section 3). If $\mathbf{M} \notin \mathcal{M}^{\text{impl}}$ and $\mathbf{M} \neq \emptyset$, the cost $J^{\mathbf{M}}$ is given by the available supermode implementations: Given an implemented supermode $\mathbf{M}' \in \mathcal{M}^{\text{impl}} \cap \overline{\mathcal{M}}(\mathbf{M})$ of $\mathbf{M}$, the cost of $\mathbf{M}$ when using the implementation of $\mathbf{M}'$ is

$$J^{\mathbf{M}}(\mathbf{M}') = \sum_{i \in \mathcal{I}_{\mathbf{M}}} J_i^{\mathbf{M}'} = J^{\mathbf{M}'} - \sum_{i \in \mathcal{I}_{\mathbf{M}'} \setminus \mathcal{I}_{\mathbf{M}}} J_i^{\mathbf{M}'}. \quad (3)$$

At runtime, we use the supermode implementation that gives the smallest cost, and thus

$$J^{\mathbf{M}} = \min_{\mathbf{M}' \in \mathcal{M}^{\text{impl}} \cap \overline{\mathcal{M}}(\mathbf{M})} J^{\mathbf{M}}(\mathbf{M}').$$

The cost of the idle mode $\emptyset$ is defined as $J^{\emptyset} = 0$. The objective is to find a set of modes $\mathcal{M}^{\text{impl}}$ and synthesize them such that their implementations can be stored in the available memory of the platform. The cost to be minimized is

$$J_{\text{tot}} = \sum_{\mathbf{M} \in \mathcal{M}^{\text{func}}} w_{\mathbf{M}} J^{\mathbf{M}}. \quad (4)$$

6. SYNTHESIS APPROACH

Our synthesis approach consists of two sequential parts. First, we synthesize implementations for a limited set of functional modes (Section 6.1). Second, we select the implementations to store under given memory constraints, and if needed, we produce virtual-mode implementations (Section 6.2).

[1]Due to the large number of modes, it can be nonpractical (or even impossible) to explicitly mark the set of functional modes. In such cases, however, the designer can indicate control loops that cannot be active in parallel due to functionality restrictions. Then, the set of functional modes $\mathcal{M}^{\text{func}}$ includes all modes except those containing two or more mutually exclusive controllers. If no specification of functional modes is made, it is assumed that $\mathcal{M}^{\text{func}} = \mathcal{M}$.

[2]It can be impossible for the designer to assign weights to all functional modes explicitly. An alternative is to assign weights to some particularly important and frequent modes (all other functional modes get a default weight). Another alternative is to correlate the weights to the occurrence frequency of modes (e.g., obtained by simulation).

Initialization:
1. $\mathcal{M}^{\text{impl}} := \emptyset$
2. **list** `edges` := **empty**

For each mode $\mathbf{M} \in \mathcal{M}^{\text{func}}_{\uparrow}$, perform Steps 3–5 below.
3. Synthesize $\mathbf{M}$ and set $\mathcal{M}^{\text{impl}} := \mathcal{M}^{\text{impl}} \cup \{\mathbf{M}\}$
4. For each $\mathbf{M}' \in \mathcal{M}^{-}(\mathbf{M})$, add the edge $(\mathbf{M}, \mathbf{M}')$ to the beginning of the list `edges`
5. **while** `edges` $\neq$ **empty**
   (a) Remove edge $(\mathbf{M}, \mathbf{M}')$ from the beginning of `edges`
   (b) **if** $\mathbf{M}' \in \mathcal{M}^{\text{virt}}$ **and** $\mathbf{M}' \neq \emptyset$, for each $\mathbf{M}'' \in \mathcal{M}^{-}(\mathbf{M}')$, add $(\mathbf{M}, \mathbf{M}'')$ to the beginning of `edges`
   (c) **else if** $\mathbf{M}' \in \mathcal{M}^{\text{func}} \setminus \mathcal{M}^{\text{impl}}$ **and** $\mathbf{M}' \neq \emptyset$
      i. Synthesize $\mathbf{M}'$ and set $\mathcal{M}^{\text{impl}} := \mathcal{M}^{\text{impl}} \cup \{\mathbf{M}'\}$
      ii. $\Delta J^{\mathbf{M}'}(\mathbf{M}) := \left(J^{\mathbf{M}'}(\mathbf{M}) - J^{\mathbf{M}'}\right) \Big/ J^{\mathbf{M}'}(\mathbf{M})$
      iii. Compute the average weight w^{avg} of the immediate functional submodes of $\mathbf{M}'$
      iv. **if** $\Delta J^{\mathbf{M}'}(\mathbf{M}) \geqslant \frac{\lambda}{w^{\text{avg}}}$, for each $\mathbf{M}'' \in \mathcal{M}^{-}(\mathbf{M}')$, add $(\mathbf{M}', \mathbf{M}'')$ to the beginning of `edges`

Figure 5: Synthesis with improvement factor λ

6.1 Control-Quality versus Synthesis Time

Our heuristic is based on a limited depth-first exploration of the Hasse diagram of modes. We use an improvement factor $\lambda \geqslant 0$ to limit the exploration and to trade quality with optimization time. Given a set of modes $\mathcal{M}' \subseteq \mathcal{M}$, let us introduce the set $\mathcal{M}'_{\uparrow} = \{\mathbf{M} \in \mathcal{M}' : \overline{\mathcal{M}}(\mathbf{M}) \cap \mathcal{M}' = \emptyset\}$, which contains the modes in $\mathcal{M}'$ that do not have supermodes in the same set $\mathcal{M}'$. Such modes are called *top* modes of $\mathcal{M}'$. For example, mode $\mathbf{\Lambda}$ is the only top mode of $\mathcal{M}$ (i.e., $\mathcal{M}_{\uparrow} = \{\mathbf{\Lambda}\}$). We introduce the set of immediate submodes of $\mathbf{M} \in \mathcal{M}$ as $\mathcal{M}^{-}(\mathbf{M}) = \{\mathbf{M}' \in \underline{\mathcal{M}}(\mathbf{M}) : |\mathbf{M}| - 1 = |\mathbf{M}'|\}$, and the set of immediate supermodes as $\mathcal{M}^{+}(\mathbf{M}) = \{\mathbf{M}' \in \overline{\mathcal{M}}(\mathbf{M}) : |\mathbf{M}| + 1 = |\mathbf{M}'|\}$—for example, $\mathcal{M}^{+}(\{\Lambda_2\}) = \{\{\Lambda_1, \Lambda_2\}, \{\Lambda_2, \Lambda_3\}\}$.

Figure 5 outlines our approach. In the first and second step, the set of modes with produced implementations is initialized to the empty set and an empty list is initialized. In this first part of the synthesis, we consider only implementations of functional modes. Virtual modes are implemented only as a solution to memory constraints (Section 6.2). Note that we must implement the top functional modes (i.e., functional modes that do not have any functional supermodes). For each such mode $\mathbf{M}$, we perform Steps 3–5. In Step 3, we run the synthesis for mode $\mathbf{M}$ to produce an implementation (periods, control laws, and schedule) [8]. After the synthesis of a mode, the set $\mathcal{M}^{\text{impl}}$ is updated. We proceed, in Step 4, by finding the edges from mode $\mathbf{M}$ to its immediate submodes. These edges are added to the beginning of the list `edges`, which contains the edges leading to modes that are chosen for synthesis.

As long as the list `edges` is not empty, we perform the substeps in Step 5. First, in Step (a), an edge $(\mathbf{M}, \mathbf{M}')$ is removed from the beginning of `edges`. In Step (b), if $\mathbf{M}' \neq \emptyset$ is a virtual mode, we do not consider it in the synthesis and resume the exploration with its immediate submodes. In Step (c), if an implementation of the functional mode $\mathbf{M}' \neq \emptyset$ has not yet been synthesized, we perform four steps (Steps i–iv). We first run the synthesis for mode $\mathbf{M}'$. Based on the obtained cost, we decide whether or not to continue synthesizing modes along the current path (i.e., synthesize immediate submodes of $\mathbf{M}'$). To take this decision, we consider the cost improvement of the implementation of mode $\mathbf{M}'$ relative to the cost when using the implementation of mode $\mathbf{M}$ to operate the system in mode $\mathbf{M}'$. We also consider the weights of the immediate submodes of $\mathbf{M}'$. In Step ii, we compute the relative cost improvement $\Delta J^{\mathbf{M}'}(\mathbf{M})$. The cost $J^{\mathbf{M}'}$ of the synthesized mode $\mathbf{M}'$ was de-

1. **if** $\sum_{\mathbf{M}\in\mathcal{M}^{\text{impl}}_{\uparrow}} \mathsf{mem}^{\mathbf{M}}_d > \mathsf{mem}^{\max}_d$ for some $d \in \mathcal{I}_{\mathbf{N}}$
   (a) Find $\mathcal{M}' \subseteq \bigcup_{\mathbf{M}\in\mathcal{M}^{\text{impl}}_{\uparrow}} \mathcal{M}^{+}(\mathbf{M})$ with smallest size such that, for each $\mathbf{M} \in \mathcal{M}^{\text{impl}}_{\uparrow}$, $\mathcal{M}' \cap \overline{\mathcal{M}}(\mathbf{M}) \neq \emptyset$
   (b) For each $\mathbf{M}' \in \mathcal{M}'$, synthesize $\mathbf{M}'$ and set $\mathcal{M}^{\text{impl}} := \mathcal{M}^{\text{impl}} \cup \{\mathbf{M}'\}$
   (c) Repeat Step 1
2. Find $b_{\mathbf{M}} \in \{0,1\}$ for each $\mathbf{M} \in \mathcal{M}^{\text{impl}}$ such that the cost in Equation 5 is minimized and the constraint in Equation 6 is satisfied for each $d \in \mathcal{I}_{\mathbf{N}}$ and $b_{\mathbf{M}} = 1$ for $\mathbf{M} \in \mathcal{M}^{\text{impl}}_{\uparrow}$

Figure 6: Mode-selection approach

fined in Section 3, whereas the cost $J^{\mathbf{M}'}(\mathbf{M})$ of mode $\mathbf{M}'$ when using the implementation of mode $\mathbf{M}$ is given in Equation 3. In Step iii, we compute the average weight of the immediate functional submodes of $\mathbf{M}'$ as

$$w^{\text{avg}} = \sum_{\mathbf{M}''\in\mathcal{M}^{-}(\mathbf{M}')\cap\mathcal{M}^{\text{func}}} w_{\mathbf{M}''} \Big/ \left|\mathcal{M}^{-}(\mathbf{M}') \cap \mathcal{M}^{\text{func}}\right|.$$

Based on the given improvement factor $\lambda \geqslant 0$, the relative improvement $\Delta J^{\mathbf{M}'}(\mathbf{M})$, and the average mode weight w^{avg}, we decide in Step iv whether to consider the immediate submodes $\mathcal{M}^{-}(\mathbf{M}')$ in the continuation of the synthesis. Note that, in this way, the submodes with larger weights are given higher priority in the synthesis. If $\mathcal{M}^{-}(\mathbf{M}') \cap \mathcal{M}^{\text{func}} = \emptyset$ (i.e., all immediate submodes are virtual), the average weight is set to $w^{\text{avg}} = \infty$, which means that all immediate submodes are added in Step iv.

The parameter $\lambda \geqslant 0$ is used to tune the exploration of the set of functional modes $\mathcal{M}^{\text{func}}$; for example, a complete synthesis of the functional modes corresponds to $\lambda = 0$. The results of this first part of the synthesis are implementations of a set of functional modes $\mathcal{M}^{\text{impl}} \subseteq \mathcal{M}^{\text{func}}$. Note that all top functional modes are synthesized, which means that the system can operate in any functional mode.

6.2 Control-Quality versus Memory Consumption

Given are implementations of functional modes $\mathcal{M}^{\text{impl}}$ as a result of the first part. We shall now discuss how to select the implementations to store such that given memory constraints are satisfied and the system can operate in any functional mode with the stored implementations. Note that the set of top functional modes $\mathcal{M}^{\text{func}}_{\uparrow}$ must have implementations in memory. These implementations can be used to operate the system in any of the other functional modes. If the implementations of the top functional modes can be stored in the available memory, we do not need to implement virtual modes. The remaining problem is to additionally select the remaining implementations to store in memory. If, however, the implementations of the top functional modes cannot be stored, we must produce implementations of virtual modes. These implementations should cover a large amount of functional modes. An implementation of a virtual supermode can replace several implementations and, therefore, save memory space but with degraded control performance in the modes with replaced implementations.

To select the mode implementations to store, we propose the two-step approach outlined in Figure 6. We first check whether the implementations of the functional top modes $\mathcal{M}^{\text{impl}}_{\uparrow} = \mathcal{M}^{\text{func}}_{\uparrow}$ can be stored in the available memory on each computation node $N_d \in \mathbf{N}$. If not, we proceed by selecting virtual modes to implement. In Step (a), we find among the immediate supermodes of the top modes $\mathcal{M}^{\text{impl}}_{\uparrow}$ (the supermodes are virtual modes) a subset $\mathcal{M}'$ with minimal size that contains a supermode for each top mode. In Step (b), we produce implementations of the chosen virtual modes $\mathcal{M}'$ and add them to the set of implemented modes $\mathcal{M}^{\text{impl}}$. The top modes of $\mathcal{M}^{\text{impl}}$ is now $\mathcal{M}'$. We go back to Step 1 and check whether the implementations of the new top modes can be stored in memory. If not, we repeat Steps (a) and (b) and consider larger virtual modes until we find a set of virtual modes that cover all functional modes and that can be implemented in the available memory. After this step, we know that the implementations of $\mathcal{M}^{\text{impl}}_{\uparrow}$ can be stored in the available memory and that they are sufficient to operate the system in any functional mode.

The second step selects mode implementations to store in the available memory such the control quality is optimized. This selection is done by solving a linear program with binary variables: Given the set of implemented modes, possibly including virtual modes from Step 1 in Figure 6, let us introduce a binary variable $b_{\mathbf{M}} \in \{0,1\}$ for each mode $\mathbf{M} \in \mathcal{M}^{\text{impl}}$. If $b_{\mathbf{M}} = 1$, it means that the implementation of $\mathbf{M}$ is selected to be stored in memory, whereas it is not stored if $b_{\mathbf{M}} = 0$. For a correct implementation of the multi-mode system, we must store the implementation of each top mode $\mathbf{M} \in \mathcal{M}^{\text{impl}}_{\uparrow}$, and thus $b_{\mathbf{M}} = 1$ for those modes. To select the other mode implementations to store, we consider the obtained cost reduction by implementing a mode. Given $\mathbf{M} \in \mathcal{M}^{\text{impl}} \setminus \mathcal{M}^{\text{impl}}_{\uparrow}$, this cost reduction is

$$\Delta J^{\mathbf{M}} = \min_{\mathbf{M}'\in\mathcal{M}^{\text{impl}}\cap\overline{\mathcal{M}}(\mathbf{M})} \Delta J^{\mathbf{M}}(\mathbf{M}'),$$

where $\Delta J^{\mathbf{M}}(\mathbf{M}')$ is given in Step ii in Figure 5. By removing a mode $\mathbf{M} \in \mathcal{M}^{\text{impl}} \setminus \mathcal{M}^{\text{impl}}_{\uparrow}$ (i.e., setting $b_{\mathbf{M}} = 0$), we lose this cost improvement. The cost to be minimized is the overall loss of cost reductions for mode implementations that are removed from $\mathcal{M}^{\text{impl}} \setminus \mathcal{M}^{\text{impl}}_{\uparrow}$. This cost is formulated as

$$\sum_{\mathbf{M}\in\mathcal{M}^{\text{impl}}\setminus\mathcal{M}^{\text{impl}}_{\uparrow}} (1 - b_{\mathbf{M}}) w_{\mathbf{M}} \Delta J^{\mathbf{M}}. \tag{5}$$

The memory consumption of an implementation of mode $\mathbf{M}$ is $\mathsf{mem}^{\mathbf{M}}_d$ on node $N_d \in \mathbf{N}$. This information is given as an output of the controller scheduling and synthesis (Section 3). The memory constraint on node $N_d \in \mathbf{N}$ is

$$\sum_{\mathbf{M}\in\mathcal{M}^{\text{impl}}} b_{\mathbf{M}} \mathsf{mem}^{\mathbf{M}}_d \leqslant \mathsf{mem}^{\max}_d, \tag{6}$$

considering a memory limit $\mathsf{mem}^{\max}_d$ on node N_d. After solving the linear program in Step 2, we store the implementations of the selected modes $\{\mathbf{M} \in \mathcal{M}^{\text{impl}} : b_{\mathbf{M}} = 1\}$ and the multi-mode system synthesis is completed.

7. EXPERIMENTAL RESULTS

We have conducted experiments to evaluate our proposed approach. We created 100 benchmarks using a database of inverted pendulums, ball and beam processes, DC servos, and harmonic oscillators [11]. Such plants are acknowledged as representatives of realistic control problems and are used extensively for experimental evaluation. We considered 10 percent of the modes to be virtual for benchmarks with 6 or more control loops. We generated 3 to 5 tasks for each control application; thus, the number of tasks in our benchmarks varies from 12 to 60. The tasks were mapped randomly to platforms comprising 2 to 10 nodes. Further, we considered uniform distributions of the execution times of tasks. The best-case and worst-case execution times of the tasks were chosen randomly from 2 to 10 milliseconds and the communication times of messages were chosen from 2 to 4 milliseconds.

The first set of experiments evaluate the synthesis heuristic in Section 6.1. The synthesis was run for each benchmark and for different values of the improvement factor λ. All experiments were run on a PC with a quad-core CPU at 2.2 GHz, 8 Gb of RAM, and running Linux. For each value of λ and for each benchmark, we computed the obtained cost after the synthesis. This cost is $J^{(\lambda)}_{\text{tot}} = \sum_{\mathbf{M}\in\mathcal{M}^{\text{func}}} w_{\mathbf{M}} J^{\mathbf{M}}$ (Equation 4) and, because of the large number of modes, we could compute

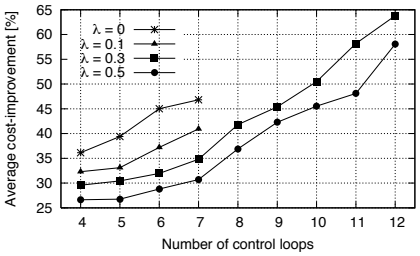

Figure 7: Control-performance improvements

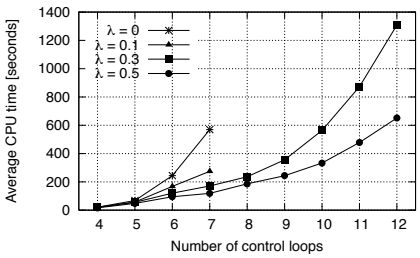

Figure 8: CPU times for mode synthesis

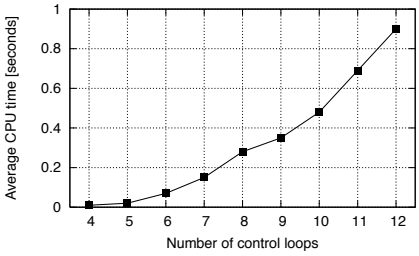

Figure 9: CPU times for mode selection ($\lambda = 0.3$)

it in affordable time for benchmarks with at most 12 control loops. It is very important to note that the time-consuming calculation of the cost is only needed for the experimental evaluation and is not part of the heuristic. We used the values 0, 0.1, 0.3, and 0.5 for the improvement factor λ. Because of long runtimes, we could run the synthesis with $\lambda = 0$ (exhaustive synthesis of all functional modes) and $\lambda = 0.1$ for benchmarks with at most 7 control loops. As a baseline for the comparison, we used the cost $J^{\uparrow}_{\text{tot}}$ obtained when synthesizing only the top functional modes $\mathcal{M}^{\text{func}}_{\uparrow}$. For each benchmark and for each value of λ, we computed the achieved cost improvement as the relative difference $\Delta J^{(\lambda)}_{\text{tot}} = \left(J^{\uparrow}_{\text{tot}} - J^{(\lambda)}_{\text{tot}}\right) \Big/ J^{\uparrow}_{\text{tot}}$. Figure 7 shows the obtained cost improvements. The vertical axis indicates the average value of the relative cost-improvement $\Delta J^{(\lambda)}_{\text{tot}}$ for benchmarks with number of control loops given on the horizontal axis. The results show that the achieved control performance becomes better with smaller values on λ (i.e., a more thorough exploration of the set of modes). We also observe that the improvement is better the larger number of control loops in the multi-mode system. This demonstrates that for large number of control loops, it is important to additionally synthesize other modes than the top functional modes. The experiments also show that good quality results can be obtained also with larger values of λ, which provide affordable runtimes for even larger examples. The runtimes for the synthesis heuristic are shown in Figure 8. We show the average runtimes in seconds for the different values of the improvement factor λ and for each dimension of the benchmarks (number of control loops). Figures 7 and 8 illustrate how the designer can trade off quality of the synthesis with optimization time, depending on the size of the control system. To further illustrate the scaling of synthesis time, we mention that a system consisting of 20 control loops on 12 nodes has been synthesized with $\lambda = 0.5$ in 38 minutes.

In the second set of experiments, we evaluate the mode-selection approach. We have run the mode-synthesis heuristic (Section 6.1) with $\lambda = 0.3$. Let us denote with $\mathsf{mem}^{\text{req}}_d$ the amount of memory needed to store the generated mode implementations on each node N_d. We have run the mode selection (Section 6.2) considering a memory limitation of $0.7\mathsf{mem}^{\text{req}}_d$ for each node. To solve the linear program given by Equations 5 and 6, we used the `eplex` library for mixed integer programming in ECL^{i}PS^e [1]. The average runtimes are shown in seconds in Figure 9. Note that the mode selection is optimal if all top functional modes can be stored in memory. The selection of modes to store in memory is performed only once as a last step of the synthesis, and as the figure shows, the optimal selection is found in neglectable runtime compared to the overall runtime of the multi-mode system synthesis. To study the quality degradation as a result of memory limitations, we used a benchmark with 12 control loops running on 10 nodes. We have run the mode synthesis for three scenarios: First, we considered no memory limitation in the platform, which resulted in a cost improvement of 48.5 percent, relative to the cost obtained with the baseline approach (synthesis of only top functional modes). Second, we considered that only 70 percent of the memory required by the produced implementations can be used. As a result of the mode selection, all top functional modes could be stored, leading to a cost improvement of 41.4 percent (a degradation of 7.1 percent compared to the first scenario without memory constraints). For the third and last scenario, we considered memory limitations such that the implementations of the top functional modes cannot be all stored in memory. After the mode-selection approach, including the synthesis of virtual modes, we obtained a solution with a cost improvement of only 30.1 percent (a degradation of 18.4 percent compared to the case without memory constraints).

8. CONCLUSIONS

We addressed control-performance optimization for embedded multi-mode control systems. The main design difficulty is raised by the potentially very large number of possible modes. In this context, an appropriate selection of the actual modes to be implemented is of critical importance. We presented our synthesis approach that produces schedules and controllers for an efficient deployment of embedded multi-mode control systems.

9. REFERENCES

[1] K. R. Apt and M. G. Wallace. *Constraint Logic Programming using ECL^{i}PS^e*. Cambridge University Press, 2007.

[2] E. Bini and A. Cervin. Delay-aware period assignment in control systems. In *Proc. of the* 29^{th} *Real-Time Systems Symposium*, pp. 291–300, 2008.

[3] R. Bosch GmbH. *CAN Specification Version 2.0*. 1991.

[4] FlexRay Consortium. *FlexRay Communications System. Protocol Specification Version 2.1*. 2005.

[5] H. Kopetz. *Real-Time Systems—Design Principles for Distributed Embedded Applications*. Kluwer Academic, 1997.

[6] B. Lincoln and A. Cervin. Jitterbug: A tool for analysis of real-time control performance. In *Proc. of the* 41^{st} *Conference on Decision and Control*, pp. 1319–1324, 2002.

[7] H. Rehbinder and M. Sanfridson. Integration of off-line scheduling and optimal control. In *Proc. of the* 12^{th} *Euromicro Conference on Real-Time Systems*, pp. 137–143, 2000.

[8] S. Samii, A. Cervin, P. Eles, and Z. Peng. Integrated scheduling and synthesis of control applications on distributed embedded systems. In *Proc. of the Design, Automation and Test in Europe Conference*, pp. 57–62, 2009.

[9] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin. On task schedulability in real-time control systems. In *Proc. of the* 17^{th} *Real-Time Systems Symposium*, pp. 13–21, 1996.

[10] K. E. Årzén, A. Cervin, J. Eker, and L. Sha. An introduction to control and scheduling co-design. In *Proc. of the* 39^{th} *Conference on Decision and Control*, pp. 4865–4870, 2000.

[11] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems*. Prentice Hall, 1997.